# インターネット計測とデータ解析 第 3 回

長 健二朗

2014 年 4 月 21 日

# 前回のおさらい

第 2 回 データとばらつき (4/14)

- 要約統計量 (平均、標準偏差、分布)
- サンプリング
- グラフによる可視化
- 演習: gnuplot によるグラフ描画

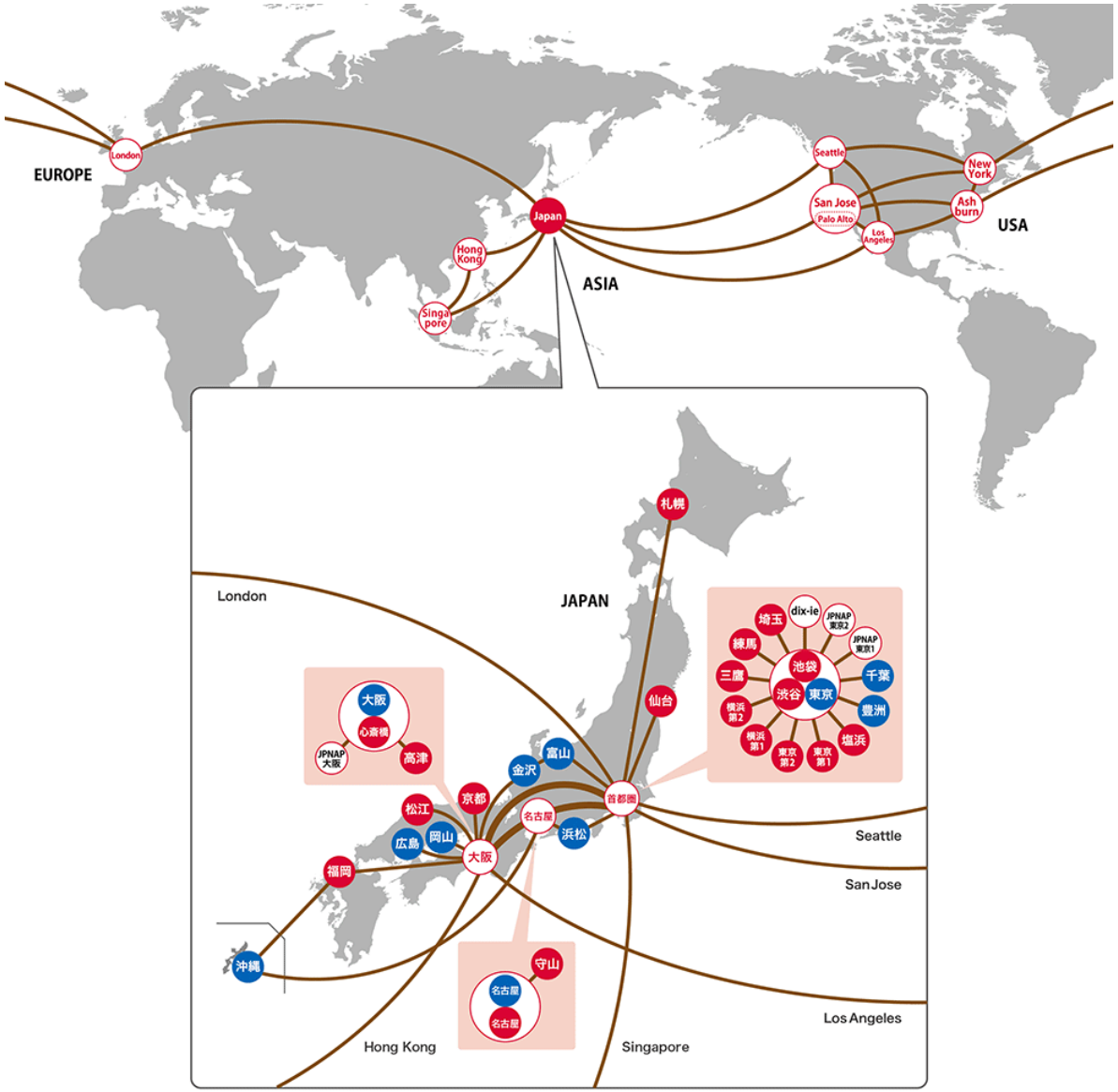
# 今日のテーマ

第 3 回 データの収集と記録

- データフォーマット
- ログ解析手法
- 演習: ログデータと正規表現

# ネットワークの構成: 日本のある ISP の場合

東京-大阪を中心に地域拠点を冗長構成で接続



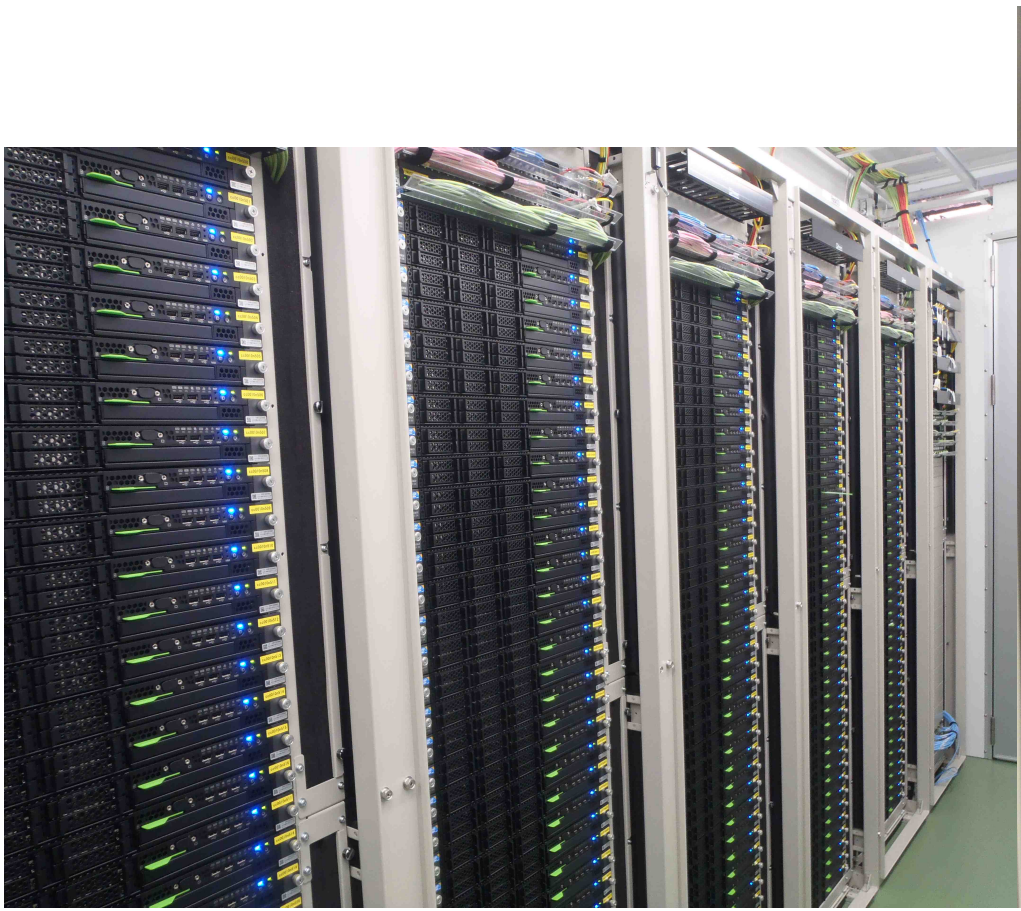
# ルータ

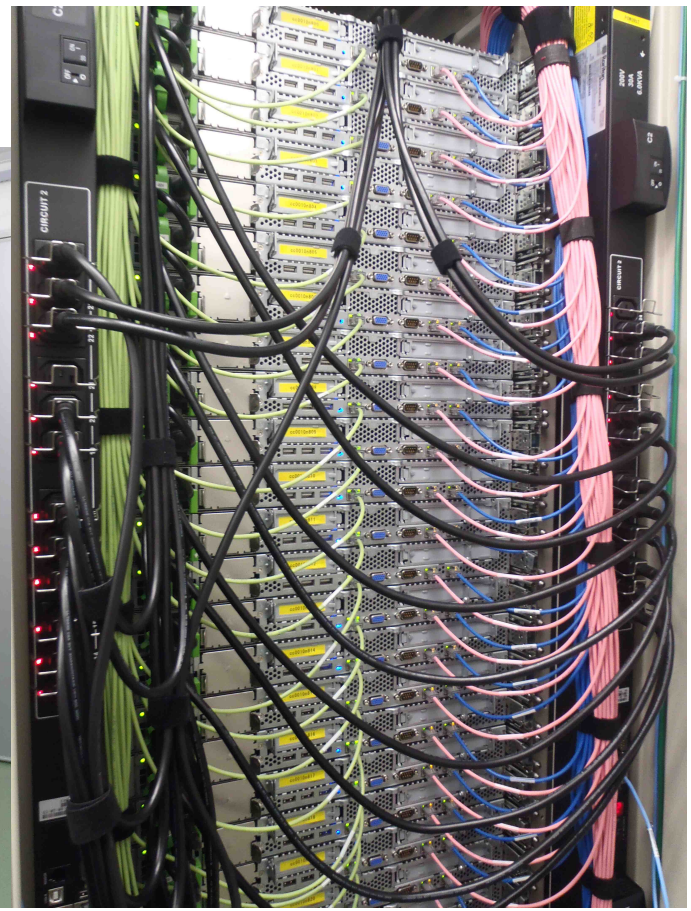
ルータ: ネットワークを接続する装置

- 機能
  - 経路制御、パケットフォワーディング、管理機能
- ルータの分類
  - コアルータ、エッジルータ、ブロードバンドルータなど

# データセンター

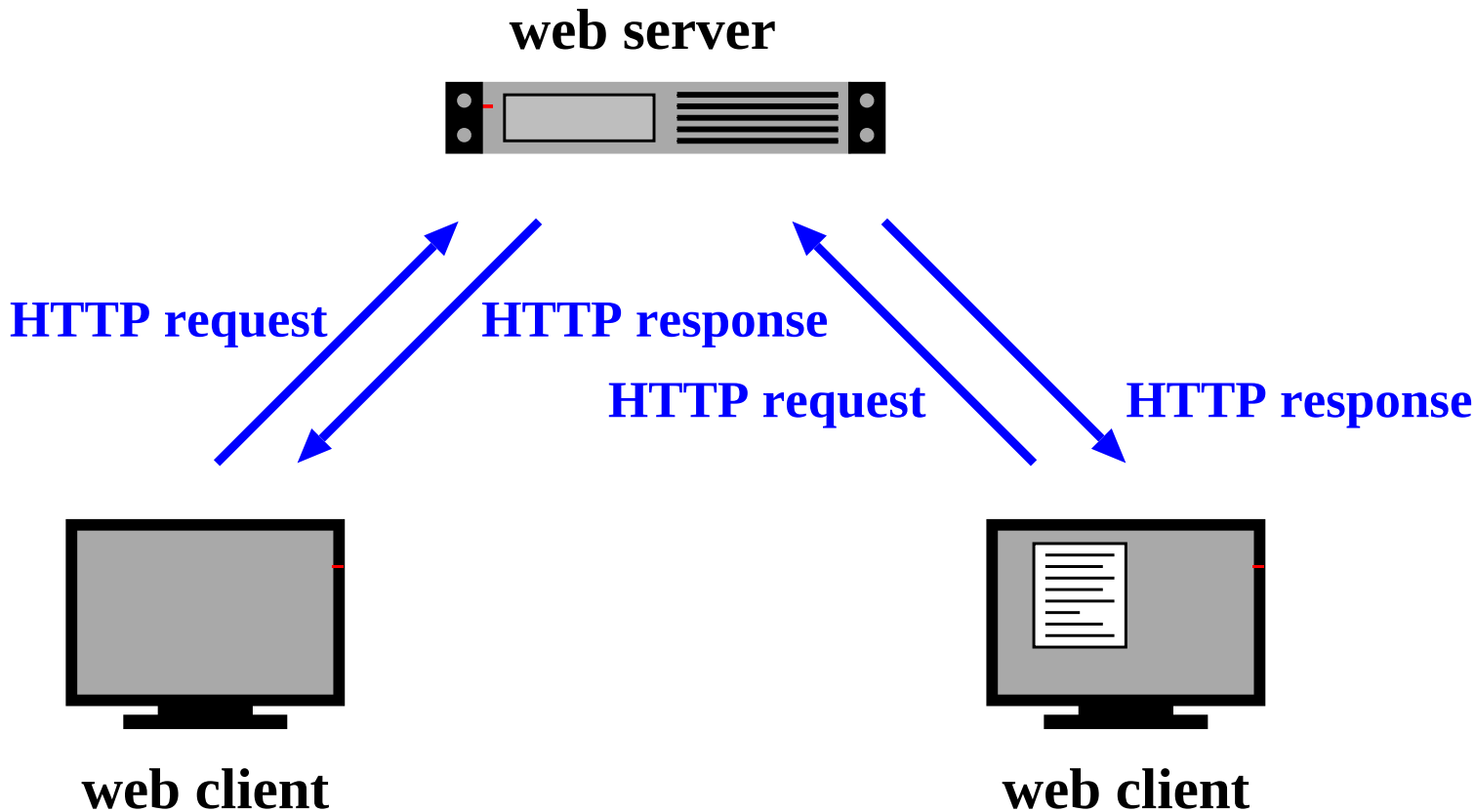
- サーバーや通信装置を収容する施設
- 給電、空調、フリーアクセス、耐震、耐災害





# Web サーバへのアクセス

- World Wide Web
  - URI: インターネット上のリソースを指定する識別子
  - HTML: Web ドキュメントを記述するマークアップ言語
  - HTTP: Web コンテンツの送受信プロトコル



# Uniform Resource Identifier (URI)

- インターネット上のリソースを指定する識別子
  - 位置や名前の指定子
  - URL (Uniform Resource Locator): 位置の識別子、URI の一部
- WWW の設計思想: あらゆる情報を指定可能にする

```
Example URIs:
      http://www.ietf.org/rfc/rfc2396.txt
      ftp://ftp.is.co.za/rfc/rfc1808.txt
      ldap://[2001:db8::7]/c=GB?objectClass?one
      mailto:John.Doe@example.com
      tel:+1-816-555-1212
      urn:oasis:names:specification:docbook:dtd:xml:4.1.2

Syntax:
        foo://example.com:8042/over/there?name=ferret#nose
        \_/   \______________/\_________/ \_________/ \__/
         |           |            |            |        |
      scheme     authority       path        query   fragment
         |   _____________________|__
        / \ /                        \
        urn:example:animal:ferret:nose
```

# HyperText Markup Language (HTML)

- Web ドキュメントを記述するマークアップ言語
  - プレインテキストの要素に付加情報を付ける
- HTML タグ: "<" と ">" で囲まれたマークアップ符合

```
<!DOCTYPE html>
<html>
  <head>
    <title>sample title</title>
  </head>
  <body>
    <h1>Heading level 1</h1>
    <h2>Heading level 2</h2>

    <p>This is a paragraph.</p>

    <p>Another paragraph with
      <a href="http://www.keio.ac.jp/">a link to Keio</a>.
    </p>
    <img src="logo.gif" alt="keio logo" width="50" height="50">
  </body>
</html>
```

# HyperText Transfer Protocol (HTTP)

- Web コンテンツの送受信プロトコル
  - TCP 上のテキストベースプロトコル

Client request:

```
GET /index.html HTTP/1.1
Host: www.example.com
Referer: http://www.example.co.jp/somepage.html
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:28.0) Gecko/20100101 Firefox/28.0
```

Server Response:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
ETag: "3f80f-1b6-3e1cb03b"
Content-Type: text/html; charset=UTF-8
Content-Length: 131
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

# データログ

- コンピュータが自動的に生成するイベントの記録
- ネットワークレベルのログ
  - 接続機器、利用 IP アドレス、パケット、通信量など
- インターネットサービスレベルのログ
  - Web アクセス、メール配送、ファイアーウォールなど
- 他の領域への広がり
  - オンラインユーザ行動、位置情報、自動車の動作記録なども

# いろいろなログ

- web server accesslog
- mail log
- syslog
- firewall log
- IDS log
- その他 あらゆる記録

# なぜログ解析をするのか？

- 現状の把握
  - 新しい発見: 技術の進歩や利用形態の変化
  - そのうえで将来予測
- セキュリティ上の問題や機器故障、それらの兆候の把握
- 解析技術の向上
  - 自動化
- 障害のレポート、問題への対応
- 記録の必要性
  - 法的理由、その他

- 利用者にカスタマイズしたサービス提供

# ログ解析の問題

- 膨大なデータ量
- 必要な情報や精度の欠如、時刻情報や内容の信憑性
- (収集システムの障害などによる) 記録の欠落
- さまざまなフォーマットが存在
- 解析には時間と労力が必要
- 解析は難しいという思い込み

- プライバシー問題

# ログの管理

- ログ収集
  - プログラミング (syslog API の利用など)
  - 収集システム構築
- ログローテーション
  - 古いデータを一定期間保存した後削除
  - ログサイズ、時間、データの古さ
  - ローテーション時にデータを失わないよう工夫
- RRD (Round Robin Database)
  - 古いログを集約することで、データサイズを一定にする
  - 例: 5 分粒度で 1 週間、2 時間粒度で 1 カ月、1 日粒度で 1 年
- 可視化
  - グラフ化して状況の把握を容易に

# さまざまなフォーマット

- web server access log
- mail log
- DHCP server log
- syslog

- テキストベースが多い、専用データベースも通常テキストへ変換可能
- 非定型データ解析

# web server access log

- Apache Common Log Format
  - client_IP client_ID user_ID time request status_code size
- Apache Combined Log Format
  - Common Log Format に referer と User-agent を追加
  - client_IP client_ID user_ID time request status_code size referer user-agent
- その他 カスタマイズ可能

client_IP: アクセス元の IP アドレス
client_ID: クライアントの識別子
user_ID: 認証ユーザ名
time: 時刻
request: リクエストの最初の行
status_code: レスポンスステータス
size: 送信バイト数（ヘッダーは含まず）0 バイトだと"-"
referer: リクエストのリンク元
user-agent: リクエスト元のブラウザの種類やバージョン

例 Combined Log Format:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] \
  "GET /apache_pb.gif HTTP/1.0" 200 2326 \
  "http://www.example.com/start.html" \
  "Mozilla/4.08 [en] (Win98; I ;Nav)"
```

# mail log

受信、送信などのメール処理毎にログが取られる
例:

```
Oct 27 13:32:54 server3 sm-mta[24510]: m9R4WsBe024510:\
 from=<client@example.com>, size=2403, class=0, nrcpts=1 \
 msgid=<201012121547.oBCFlPX6032787@example.com>, \
 proto=ESMTP, daemon=MTA, relay=mail.example.co.jp [192.0.2.1] \
Oct 27 14:43:04 server3 sm-mta[24511]: m9R4WsBe024510: \
 to=<user@example.co.jp>, delay=01:10:10 xdelay=00:00:00, \
 mailer=local, pri=32599, dsn=2.0.0, stat=Sent
```

- 時刻
- ホスト名
- プロセスオーナー [プロセス番号]
- Queue ID: メールの内部 ID
- ...
- nrcpts: 受信者数
- relay: 次の送信先サーバ
- dsn: Delivery Status Notification, RFC3463
  - 2.X.X:Success, 4.X.X:Persistent Transient Failure, 5.X.X:Permanent Failure
- stat: Message Status
  - Sent, Deferred, Bounced, etc

# DHCP server log

SYSLOG: メッセージの記録

```
Oct 28 15:04:32 server33 dhcpd: DHCPDISCOVER from 00:23:df:ff:a8:a7 via eth0
Oct 28 15:04:32 server33 dhcpd: DHCPOFFER on 192.168.2.101 \
  to 00:23:df:ff:a8:a7 via eth0
Oct 28 15:04:32 server33 dhcpd: DHCPREQUEST for 192.168.2.101 \
  from 00:23:df:ff:a8:a7 via eth0
Oct 28 15:04:32 server33 dhcpd: DHCPACK on 192.168.2.101 \
  to 00:23:df:ff:a8:a7 via eth0
Oct 28 15:09:32 server33 dhcpd: DHCPREQUEST for 192.168.2.101 \
  from 00:23:df:ff:a8:a7 via eth0
Oct 28 15:09:32 server33 dhcpd: DHCPACK on 192.168.2.101 \
  to 00:23:df:ff:a8:a7 via eth0
```

dhcpd.leases: 割り当てた IP アドレスの個別情報

```
lease 192.168.100.161 {
  starts 4 2010/12/09 23:13:39;
  ends 5 2010/12/10 00:13:39;
  tstp 5 2010/12/10 00:13:39;
  binding state free;
  hardware ethernet 5c:26:0a:17:06:00;
}
```

# syslog

- UNIX 系 OS で任意のメッセージを通知したり保存する仕組み
  - もともとメールの処理記録保存用だったが広く使われるようになった
  - 他のサーバに転送も可能
  - ログのローテーション機能のサポート
- Windows Event Log

# Web クローラ/スクレイパー

クローラ (crawler) によるデータの収集

- クローラ: 自動でデータを収集するプログラム
- web クローラ: web ページを自動巡回する
  - 検索用データベースやインデックスの作成など
  - ページ内のリンクを辿って次のページへ
- いろいろツールが存在
  - Ruby の Mechanize など
  - 急激なアクセスは攻撃と思われるので注意

スクレイパー

- HTML を解析して、必要な情報を抜き出す
- いろいろなツールが存在
  - Ruby の Nokogiri など

# ログ解析手法

- 思いつくことを色々試す、グラフ化する
  - 手を動かしている内に分かる事、思いつく事が多い
- 処理スクリプトとコマンドラインツール (grep, sort, uniq, sed, awk, etc)
- 大量データを効率よく処理する工夫
- 繰り返し行う処理はできるだけ自動化する
  - いっぽうで自動化した処理を過信しないこと

# 大量データの扱い

- ナイーブにやると膨大なデータの読み込みや処理テーブルが必要
  - データ構造やアルゴリズムを勉強しておくと役立つ
- 大量データを扱う工夫
  - 集計に不要な情報の削除
  - 時間的、空間的に集約
  - 必要に応じて分割処理
  - 必要に応じて分散並列処理
- 中間ファイルに変換する、分割処理する
- 処理に必要なメモリ量の見積り
  - データ構造を工夫する
  - 一度に処理するサイズ、次元を押える工夫
- 全体の処理時間の見積り
  - 小さいデータセットで試行
  - スケールするアルゴリズム
- メモリサイズと処理時間のトレードオフに配慮

# 正規表現 (regular expression)

正規表現

- 文字列パターンの表記法、文字列の検索や置換に利用
- もともとは形式言語理論において正規言語を表すための手段
- 文字列のパターンマッチのための記法として普及
  - grep, expr, awk, vi, lex, perl, ruby, ...

Ruby の正規表現

```
Regexp class
regular expression literal: /regexp/opt
=~ operator: subject =~ /regexp/
match() method:  /regexp/.match(subject)
string class: string.match(/regexp/)
```

# Ruby の正規表現クイックレファレンス

```
[abc]  A single character: a, b or c
[^abc]  Any single character but a, b, or c
[a-z]  Any single character in the range a-z
[a-zA-Z]  Any single character in the range a-z or A-Z
^  Start of line
$  End of line
\A  Start of string
\z  End of string
.  Any single character
\s  Any whitespace character
\S  Any non-whitespace character
\d  Any digit
\D  Any non-digit
\w  Any word character (letter, number, underscore)
\W  Any non-word character
\b  Any word boundary character
(...)  Capture everything enclosed
(a|b)  a or b
a?  Zero or one of a
a*  Zero or more of a
a+  One or more of a
a{3}  Exactly 3 of a
a{3,}  3 or more of a
a{3,6}  Between 3 and 6 of a
```

# Ruby の正規表現クイックレファレンス (つづき)

```
options:
i case insensitive
m make dot match newlines
x ignore whitespace in regex
o perform #{...} substitutions only once
```

## 最長マッチと最短マッチ (最短マッチの方が高速)

```
"*"や"+"は最長マッチ、"*?"や"+?"は最短マッチ
/<.*>/.match("<a><b><c>")  # => "<a><b><c>"
/<.*?>/.match("<a><b><c>") # => "<a>"
```

# 前回の演習: 要約統計量の計算

- 平均
- 標準偏差
- 中央値

- 市民マラソンのデータを使う: 出典 P. K. Janert “Gnuplot in Action”

`http://web.sfc.keio.ac.jp/~kjc/classes/sfc2014s-measurement/marathon.txt`

# 前回の演習: 平均の計算

- 各行から、完走時間 (分) と人数を読み合計、最後に総数で割る

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

sum = 0 # sum of data
n = 0 # the number of data
ARGF.each_line do |line|
    if re.match(line)
        min = $1.to_i
        cnt = $2.to_i
        sum += min * cnt
        n += cnt
    end
end

mean = Float(sum) / n

printf "n:%d mean:%.1f\n", n, mean
```

```
% ruby mean.rb marathon.txt
n:2355 mean:171.3
```

# 前回の演習: 標準偏差の計算

- アルゴリズム: $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

data = Array.new
sum = 0 # sum of data
n = 0 # the number of data
ARGF.each_line do |line|
    if re.match(line)
        min = $1.to_i
        cnt = $2.to_i
        sum += min * cnt
        n += cnt
        for i in 1 .. cnt
            data.push min
        end
    end
end
mean = Float(sum) / n
sqsum = 0.0
data.each do |i|
    sqsum += (i - mean)**2
end
var = sqsum / n
stddev = Math.sqrt(var)
printf "n:%d mean:%.1f variance:%.1f stddev:%.1f\n", n, mean, var, stddev
```

```
% ruby stddev.rb marathon.txt
n:2355 mean:171.3 variance:199.9 stddev:14.1
```

# 前回の演習: 標準偏差の計算の改良

- アルゴリズムを改良: $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2$

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

sum = 0 # sum of data
n = 0 # the number of data
sqsum = 0 # su of squares
ARGF.each_line do |line|
    if re.match(line)
        min = $1.to_i
        cnt = $2.to_i
        sum += min * cnt
        n += cnt
        sqsum += min**2 * cnt
    end
end

mean = Float(sum) / n
var = Float(sqsum) / n - mean**2
stddev = Math.sqrt(var)

printf "n:%d mean:%.1f variance:%.1f stddev:%.1f\n", n, mean, var, stddev
```

```
% ruby stddev2.rb marathon.txt
n:2355 mean:171.3 variance:199.9 stddev:14.1
```

# 前回の演習: 中央値の計算

- 各走者のタイムを配列に入れソート、中央値を取り出す

```
# regular expression to read minutes and count
re = /^(\d+)\s+(\d+)/

data = Array.new

ARGF.each_line do |line|
    if re.match(line)
        min = $1.to_i
        cnt = $2.to_i
        for i in 1 .. cnt
            data.push min
        end
    end
end

data.sort! # just in case data is not sorted

n = data.length # number of array elements
r = n / 2 # when n is odd, n/2 is rounded down
if n % 2 != 0
    median = data[r]
else
    median = (data[r - 1] + data[r])/2
end

printf "r:%d median:%d\n", r, median
```

```
% ruby median.rb marathon.txt
r:1177 median:176
```

# 前回の演習: gnuplot

- gnuplot を使って簡単なグラフを書く

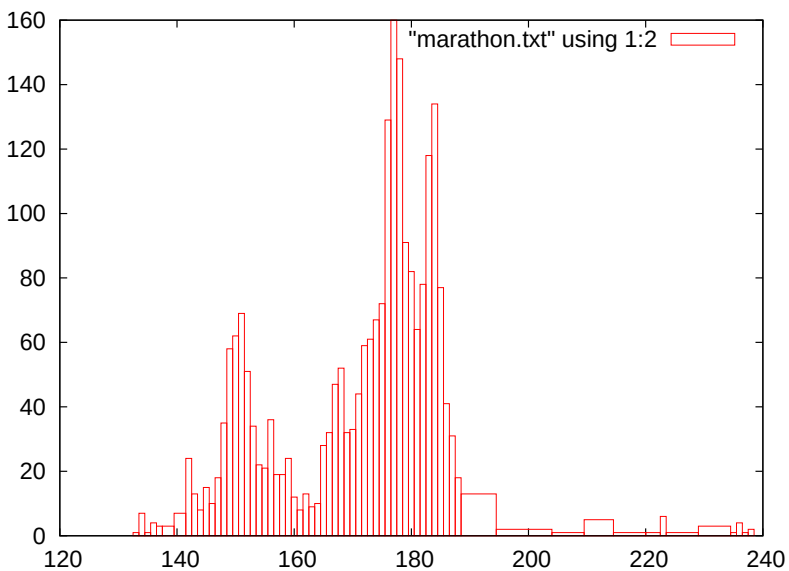
# ヒストグラム

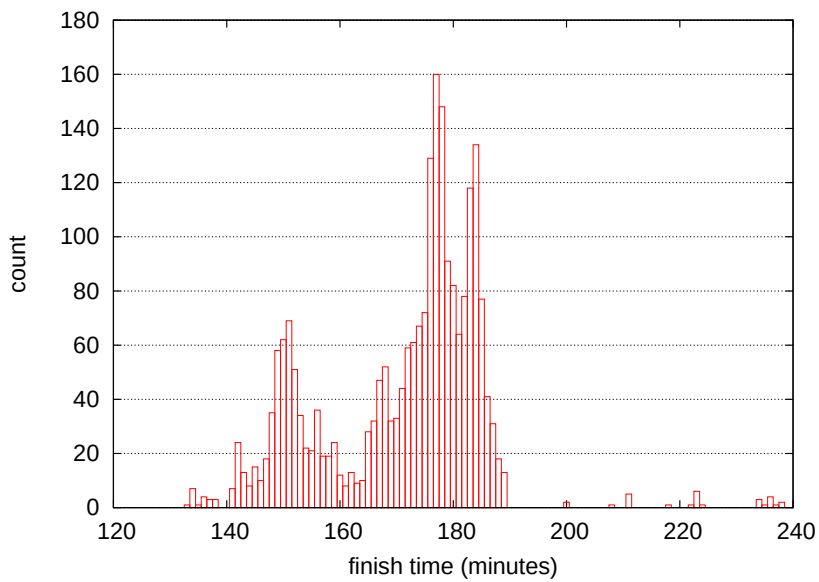
- 市民マラソンの完走タイムの分布

```
plot "marathon.txt" using 1:2 with boxes
```

## グラフを見やすくする (右側)

```
set boxwidth 1
set xlabel "finish time (minutes)"
set ylabel "count"
set yrange [0:180]
set grid y
plot "marathon.txt" using 1:2 with boxes notitle
```



# 前回の演習: 完走時間の CDF の作成

元データ:

```
# Minutes Count
133 1
134 7
135 1
136 4
137 3
138 3
141 7
142 24
...
```

累積度数を追加:

```
# Minutes Count CumulativeCount
133 1 1
134 7 8
135 1 9
136 4 13
137 3 16
138 3 19
141 7 26
142 24 50
...
```

# 前回の演習: CDF (2)

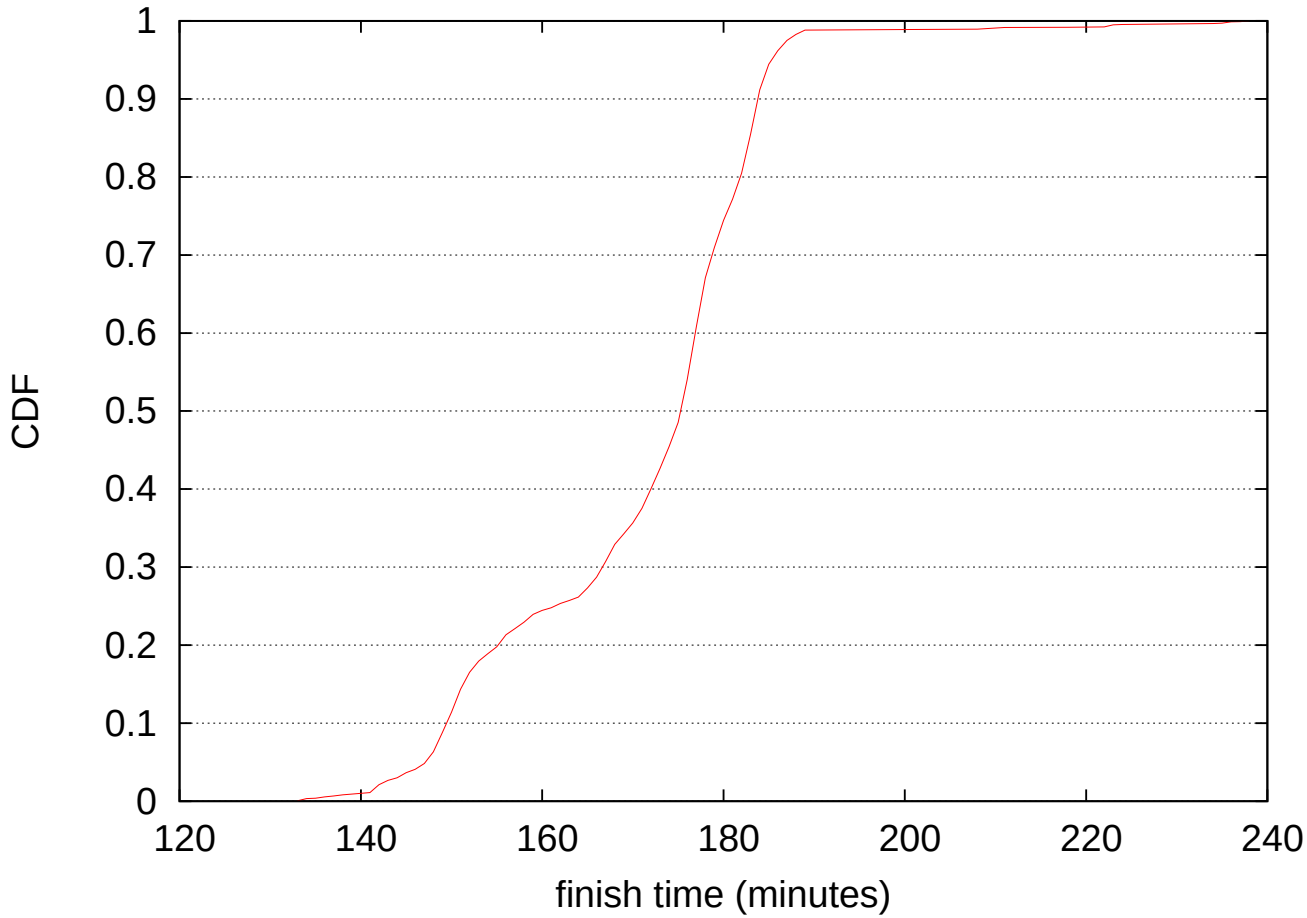
ruby code:

```
re = /^(\d+)\s+(\d+)/
cum = 0
ARGF.each_line do |line|
    begin
        if re.match(line)
            # matched
            time, cnt = $~.captures
            cum += cnt.to_i
            puts "#{time}\t#{cnt}\t#{cum}"
        end
    end
end
```

gnuplot command:

```
set xlabel "finish time (minutes)"
set ylabel "CDF"
set grid y
plot "marathon-cdf.txt" using 1:($3 / 2355) with lines notitle
```

# 市民マラソンの完走時間 CDF



# 前回の演習: プロットを画像ファイルにして保存

フォーマットを指定してファイルに保存

```
gnuplot> set terminal png
gnuplot> set output "plotfile.png"
gnuplot> replot
```

終了する時は、

```
gnuplot> quit
```

# 今日の演習: web アクセスログ サンプルデータ

- apache log (combined log format)
- JAIST のサーバーログ (24 時間分)
  - ソフトウェア配布サーバ、通常の web サーバーではない
- 1/10 サンプリング、約 72 万行
- 約 20MB (圧縮時)、約 162MB (解凍後)
- クライアントの IP アドレスは、プライバシー保護のため匿名化
  - using “ipv6loganon –anonymize-careful”

サンプルデータ:
`http://www.iijlab.net/~kjc/classes/sfc2014s-measurement/sample_access_log.zip`

# サンプルデータ

```
117.136.16.0 - - [01/Oct/2013:23:59:58 +0900] "GET /project/morefont/liangqiushengshufaziti.apk \
  HTTP/1.1" 200 524600 "-" "-" jaist.dl.sourceforge.net
218.234.160.0 - - [01/Oct/2013:23:59:59 +0900] "GET /pub/Linux/linuxmint/packages/dists/olivia/\
  upstream/i18n/Translation-ko.xz HTTP/1.1" 404 564 "-" "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" \
  ftp.jaist.ac.jp
119.80.32.0 - - [01/Oct/2013:23:59:59 +0900] "GET /project/morefont/xiongtuti.apk HTTP/1.1" 304 \
  132 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Foxy/1; InfoPath.1)" \
  jaist.dl.sourceforge.net
218.234.160.0 - - [02/Oct/2013:00:00:00 +0900] "GET /pub/Linux/linuxmint/packages/dists/olivia/\
  import/i18n/Translation-en.gz HTTP/1.1" 404 562 "-" "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" \
  ftp.jaist.ac.jp
117.136.0.0 - - [02/Oct/2013:00:00:00 +0900] "GET /project/morefont/xiaoqingwaziti.apk HTTP/1.1"\
  200 590136 "-" "-" jaist.dl.sourceforge.net
123.224.224.0 - - [02/Oct/2013:00:00:00 +0900] "GET /pub/Linux/ubuntu/dists/raring/main/i18n/\
  Translation-en.bz2 HTTP/1.1" 304 187 "-" "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" ftp.jaist.ac.jp
123.224.224.0 - - [02/Oct/2013:00:00:00 +0900] "GET /pub/Linux/ubuntu/dists/raring/multiverse/\
  i18n/Translation-en.bz2 HTTP/1.1" 304 186 "-" "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" \
  ftp.jaist.ac.jp
124.41.64.0 - - [01/Oct/2013:23:59:58 +0900] "GET /ubuntu/pool/universe/s/shorewall6/\
  shorewall6_4.4.26.1-1_all.deb HTTP/1.1" 200 435975 "-" "Wget/1.14 (linux-gnu)" ftp.jaist.ac.jp
...
240b:10:c140:a909:a949:4291:c02d:5d13 - - [02/Oct/2013:00:00:01 +0900] "GET /ubuntu/pool/main/m/\
  manpages/manpages_3.52-1ubuntu1_all.deb HTTP/1.1" 200 626951 "-" \
  "Debian APT-HTTP/1.3 (0.9.7.7ubuntu4)" ftp.jaist.ac.jp
...
```

# 今日の演習: リクエスト推移のプロット

- サンプルデータを使用
- リクエスト数と転送バイト数を 5 分間ビンで抽出する
- 結果のプロット

```
% ruby parse_accesslog.rb sample_access_log > access-5min.txt
% more access-5min.txt
2013-10-01T20:00 1 1444348221
...
2013-10-01T23:55 215 1204698404
2013-10-02T00:00 2410 5607857319
2013-10-02T00:05 2344 3528532804
2013-10-02T00:10 2502 4354264670
2013-10-02T00:15 2555 5441105487
...
% gnuplot
gnuplot> load 'access.plt'
```

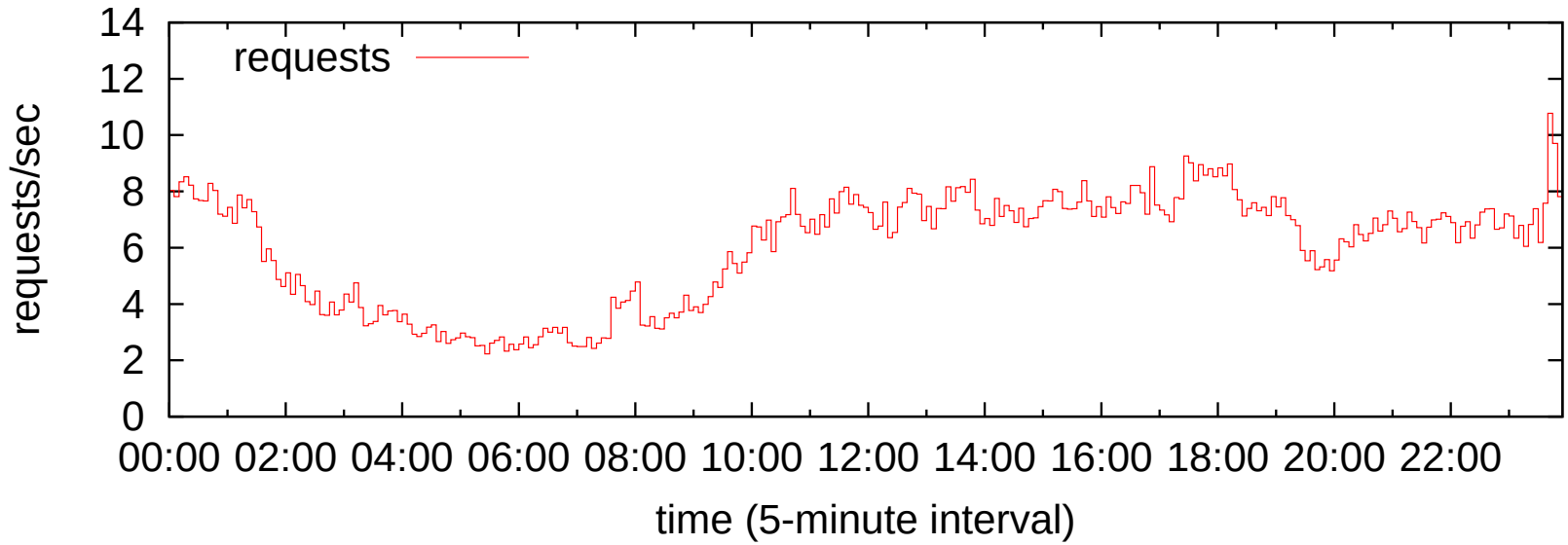
# 5 分間隔でリクエスト数と転送バイト数を抽出

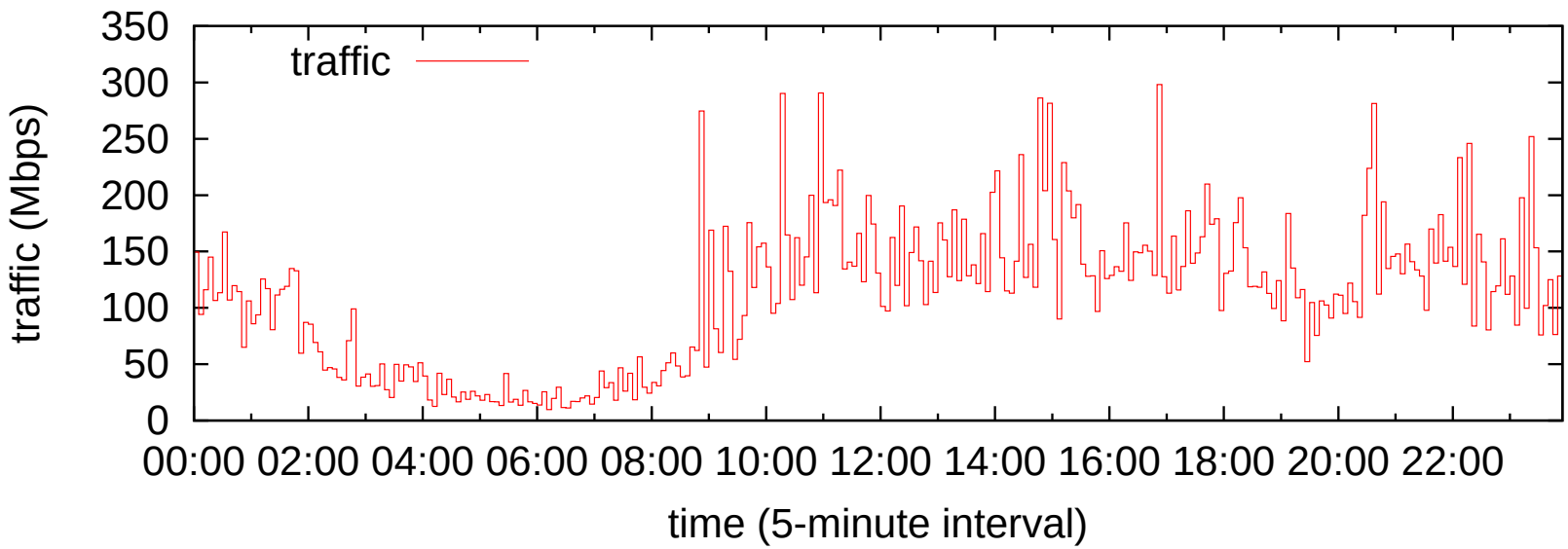
```
#!/usr/bin/env ruby
require 'date'

# regular expression for apache common log format
#   host ident user time request status bytes
re = /^(\S+) (\S+) (\S+) \[(.*?)\] "(.*?)" (\d+) (\d+|-)/
timebins = Hash.new([0, 0])
count = parsed = 0
ARGF.each_line do |line|
  count += 1
  if re.match(line)
    host, ident, user, time, request, status, bytes = $~.captures

    next unless request.match(/GET\s.*/) # ignore if the request is not "GET"
    next unless status.match(/2\d{2}/) # ignore if the status is not success (2xx)
    parsed += 1
    # parse timestamp
    ts = DateTime.strptime(time, '%d/%b/%Y:%H:%M:%S')
    # create the corresponding key for 5-minutes timebins
    rounded = sprintf("%02d", ts.min.to_i / 5 * 5)
    key = ts.strftime("%Y-%m-%dT%H:#{rounded}")
    # count by request and byte
    timebins[key] = [timebins[key][0] + 1, timebins[key][1] + bytes.to_i]
  else
    # match failed
    $stderr.puts("match failed at line #{count}: #{line.dump}")
  end
end
timebins.sort.each do |key, value|
  puts "#{key} #{value[0]} #{value[1]}"
end
$stderr.puts "parsed:#{parsed} ignored:#{count - parsed}"
```

# リクエスト推移のプロット



# gnuplot スクリプト

- multiplot 機能で 2 つのプロットをまとめる

```
set xlabel "time (5-minute interval)"
set xdata time
set format x "%H:%M"
set timefmt "%Y-%m-%dT%H:%M"
set xrange ['2013-10-02T00:00':'2013-10-02T23:55']
set key left top

set multiplot layout 2,1

set yrange [0:14]
set ylabel "requests/sec"
plot "access-5min.txt" using 1:($2/300) title 'requests' with steps

set yrange [0:350]
set ylabel "traffic (Mbps)"
plot "access-5min.txt" using 1:($3*8/300/1000000) title 'traffic' with steps

unset multiplot
```

# おまけ: 便利な UNIX コマンド

- テキストファイルを扱う場合に便利な UNIX コマンド
  - sort, head, tail, cat, cut
  - diff, tee, grep, uniq, wc
  - join, find, sed, awk, screen

- Windows の場合は、Gow (Gnu on Windows) などをインストールする必要

# sort コマンド

sort コマンド: テキストファイルの行をソートして並び替える

```
$ sort [options] [FILE ...]
```

- options (課題で使いそうなオプション)
  - -n : フィールドを数値として評価
  - -r : 結果を逆順に並べる
  - -k POS1[,POS2] : ソートするフィールド番号 (1 オリジン) を指定する
  - -t SEP : フィールドセパレータを指定する
  - -m : 既にソートされたファイルをマージする
  - -T DIR : 一時ファイルのディレクトリを指定する

例: file を第 3 フィールドを数値とみて逆順にソート、一時ファイルは”/usr/tmp”に作る

```
$ sort -nr -k3,3 -T/usr/tmp file
```

# head コマンド

head コマンド: ファイルの先頭部分を出力

- デフォルトは 10 行表示

```
head [-n lines | -c bytes] [file ...]
```

例:

```
$ sort -nr -k3,3 file | head -n 10
```

# tail コマンド

tail コマンド: ファイルの末尾部分を出力

- デフォルトは 10 行表示

```
tail [-F | -f | -r] [-q] [-b number | -c number | -n number] [file ...]
```

- 良く使うオプション
  - -f : ファイルを監視して末尾に追加された部分を出力

例:

```
monitor a log file:
$ tail -f /var/log/httpd-access.log
```

# cat コマンド

cat コマンド: ファイルの内容を (連結して) 出力

```
cat [-benstuv] [file ...]
```

例:

```
$ cat file1 file2 > file3
```

# cut コマンド

cut コマンド: 指定したファイルの各行から、それぞれの一部分を切り出して出力

```
cut -b list [-n] [file ...],
cut -c list [file ...],
cut -f list [-s] [-d delim] [file ...]
```

- 良く使うオプション
  - -b BYTE-LIST：バイト位置を指定
  - -c CHAR-LIST：文字位置を指定
  - -f FIELD-LIST：フィールド位置を指定
  - -d DELIM：フィールドの区切り文字を指定

例:

```
extract users' login names and shells from the system passwd file:
$ cut -d : -f 1,7 /etc/passwd
show the names and login times of the currently logged in users:
$ who | cut -c 1-16,26-38
```

# diff コマンド

diff コマンド: 複数ファイルを行ごとに比較し、異なる行を表示

```
diff [OPTION]... FILES
```

- 良く使うオプション
  - -u : unified diff format で出力

例:

```
$ diff -u file1 file2
```

# tee コマンド

tee コマンド: 標準入力から読んだ内容を、標準出力とファイルの両方に出力

```
tee [-ai] [file ...]
```

例:

```
$ ls | tee output.txt
```

# grep コマンド

grep コマンド: PATTERN に指定した文字列を含む行を出力する

```
grep [options] PATTERN [FILE...]
grep [options] [-e PATTERN | -f FILE] [FILE...]
```

例:

```
search lines including 'abc':
$ grep 'abc' file
count the number of lines starting with 'abc':
$ grep -c '^abc' file
```

# uniq コマンド

uniq コマンド: ソートされたファイルから重複行を削除

```
uniq [-c | -d | -u] [-i] [-f num] [-s chars] [input_file [output_file]]
```

- 良く使うオプション
  - -d : 重複行のみ表示

例:

```
$ cat file1 file2 | sort | uniq > file3
```

```
$ sort file | uniq -d
```

# wc コマンド

## wc コマンド: ファイルの行数、単語数、バイト数を出力

```
wc [-Lclmw] [file ...]
```

# join コマンド

join コマンド: 共通のフィールドでソートされたファイルの各行を結合する

```
join [-a file_number | -v file_number] [-e string] [-o list] [-t char]
    [-1 field] [-2 field] file1 file2
```

例:

```
$ cat file1
1001    orange
1002    apple
1003    grape
$ cat file2
1001    400
1002    250
1004    500
$ join file1 file2
1001 orange 400
1002 apple 250
$ join -a1 -a2 -e NULL -o '0,1.2,2.2' file1 file2
1001 orange 400
1002 apple 250
1003 grape NULL
1004 NULL 500
```

# find コマンド

## find コマンド: ファイルの検索

```
find [-H | -L | -P] [-EXdsx] [-f pathname] pathname ... expression
find [-H | -L | -P] [-EXdsx] -f pathname [pathname ...] expression
```

例:

```
print files with ".rej" suffix:
$ find . -name "*.rej" -print

print ".o" files older than 1 year
$ find . -name "*.o" -mtime +365 -print

remove empty files:
$ find . -empty -exec rm {} \;
```

# sed コマンド (streaming editor)

sed コマンド:

```
sed [-Ealn] command [file ...]
sed [-Ealn] [-e command] [-f command_file] [-I extension]
    [-i extension] [file ...]
```

- 良く使うオプション
  - -e command : コマンドの追加
  - -f command_file : 指定したファイルに記述されているコマンドを追加

例:

```
replace "old" by "new":
$ echo "old songs in old books" | sed 's/old/new/g'

print line 3-5:
$ sed -n '3,5p' file
```

# awk コマンド

awk コマンド:

- 演算機能を持つプログラム言語
- 1 行プログラムが書き易い

```
awk [ -F fs ] [ -v var=value ] [ 'prog' | -f progfile ] [ file ...  ]
```

例:

```
swap column1 and colimn2 and add sum to column3:
$ echo "12 56" | awk '{print $2,$1,$1+$2}'

extract the capacity in percent from the df command:
$ df | awk 'match($0, /[0-9]+%/) {print substr($0, RSTART, RLENGTH - 1)}'
```

# screen コマンド

screen コマンド: 仮想端末 (標準コマンドでないので、インストールする必要あり)

- ひとつのターミナルで複数の仮想ターミナルを使用できる
- 端末を切り離す機能がある
  - シェルで処理を実行中に端末を切り離しても処理はバックグラウンドで継続し、後で再接続すればもとのシェルに戻れる
    - screen : screen の起動
    - ”ctrl-a d” : detach 端末の切り離し
    - screen -r : 切り離されている端末の接続

# まとめ

第 3 回 データの収集と記録

- データフォーマット
- ログ解析手法
- 演習: ログデータと正規表現

# 次回予定

第 4 回 分布と信頼区間 (4/28)

- 正規分布
- 信頼区間と検定
- 分布の生成
- 演習: 信頼区間
- 課題 **1**